SOFTWARE QUALITY ASSURANCE & TESTING (CSE-598)

GRADUATE PROJECT REPORT

TEAM MEMBERS: Mohit Motiani & Basavaprasad Chandu

TOPIC: SATM using MVC architecture.

OBJECTIVE:

Designing and implementing a simple automatic teller machine using MVC (Model-View-Controller) Architecture and testing model, view and controller using Junit framework

INTRODUCTION:

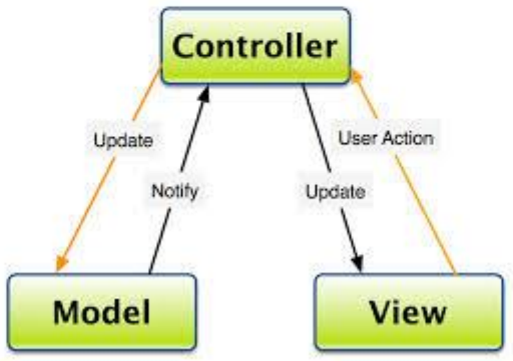
Model-View-Controller Architecture:

The MVC pattern differentiates the modelling of a pattern, its presentation and the actions based on user input into three separate classes.

Model: A model class represents the information or data of an application and the rules to manipulate the data.

View: View represents the user interface of your application. Views handle the job by providing data to the web browser or other tool that is used to make requests from your application.

Controller: The controller provide the "flow" between models and views. Controllers are responsible for processing the incoming requests from the web browser, interrogating the models for data and passing that data on to the views for presentation.



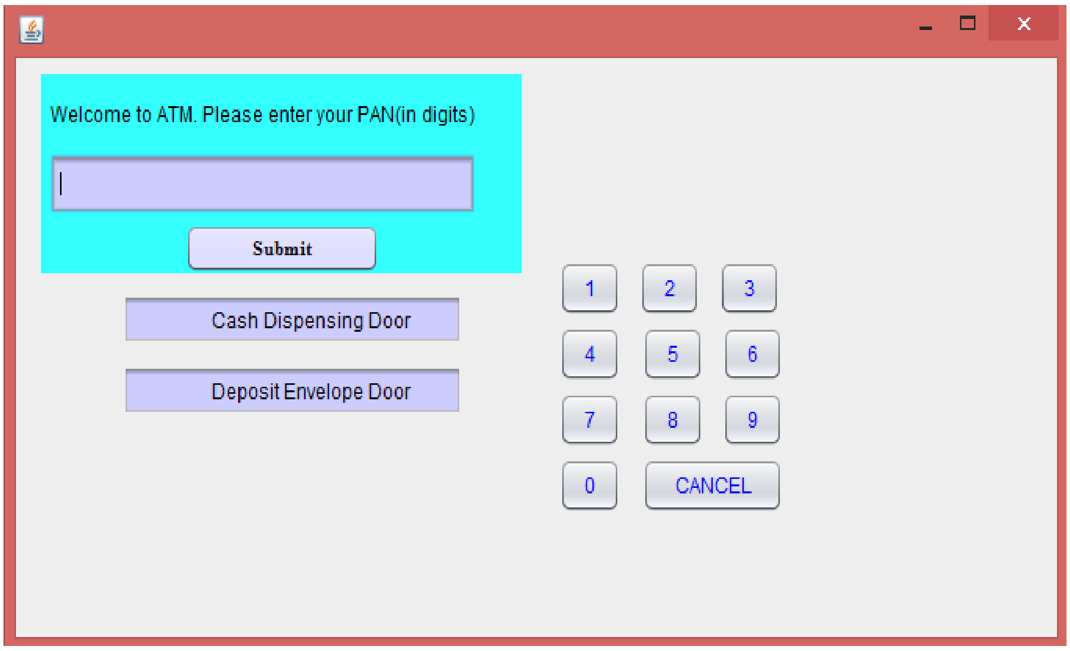
SATM APPLICATION:

SATM is a simple automatic teller machine to communicate with the bank customers. The customer can perform any of three transaction types: deposit, withdrawals and balance inquires.

Customer can exit the system at any point during the transaction. MVC architecture will be used to develop the system. View part will be done by the GUI screens for interaction with the customer. Files will be used to handle the Model (data) part of the architecture. Controller will handle the transactions as well as the updation of model at times.

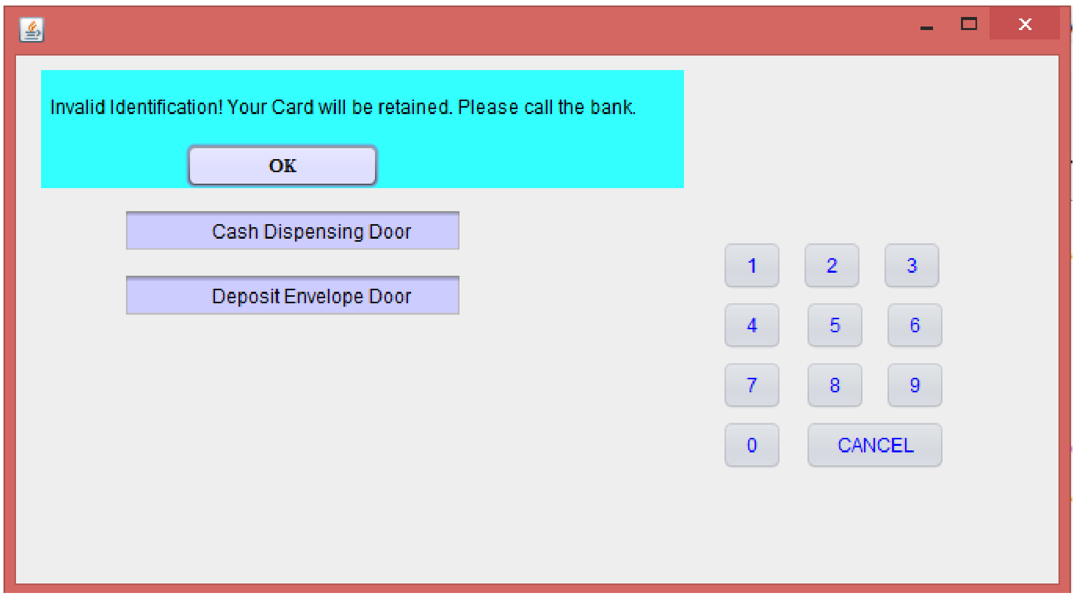
IMPLEMENTATION OF SATM APPLICATION:

- In below screen, customer is asked to enter his PAN.



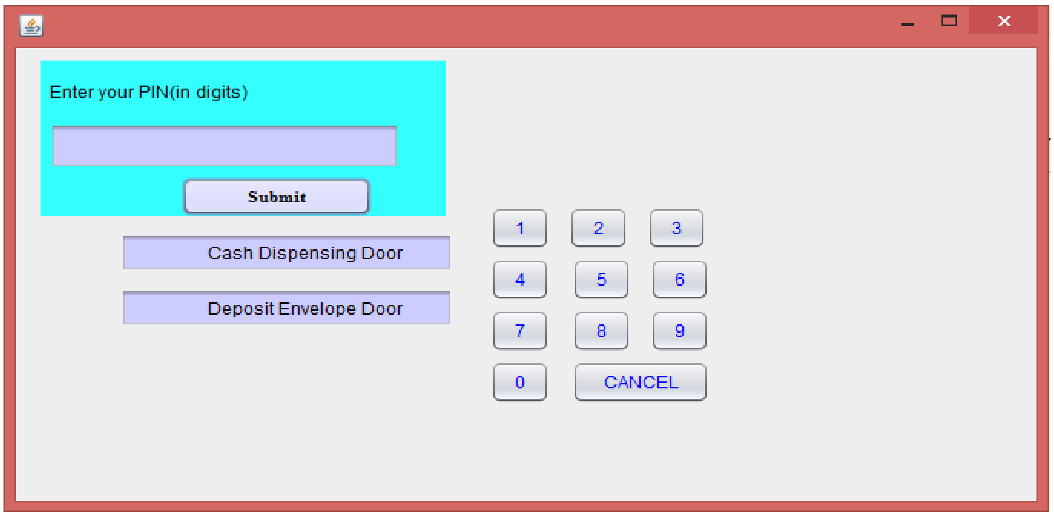
Welcome to ATM. Please enter your PAN(in digits)

- If the entered PAN is wrong, below screen will be displayed.

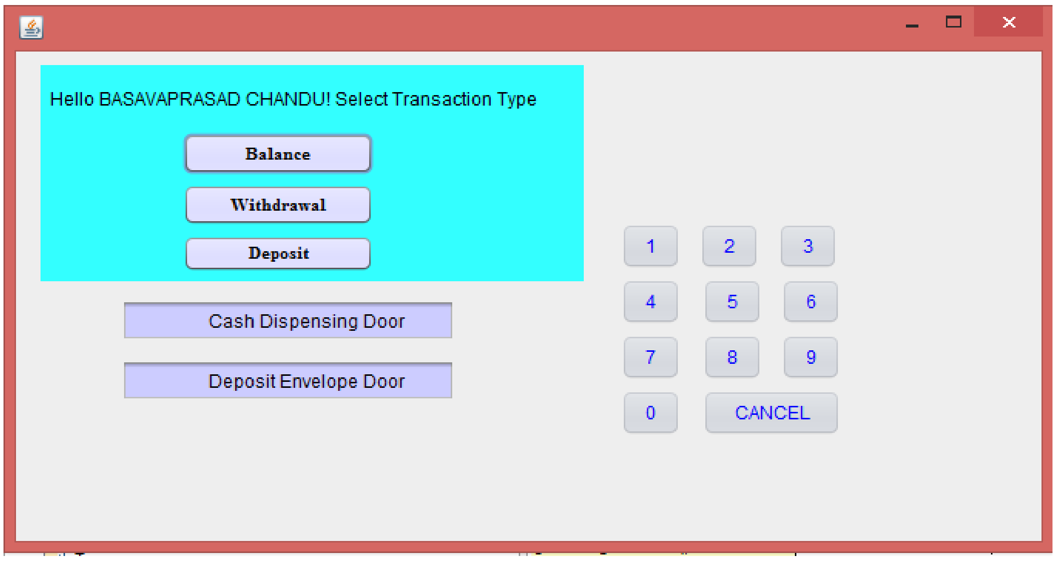


Invalid Identification! Your Card will be retained. Please call the bank.

- If the entered PAN is correct, it will take him/her to the next page.

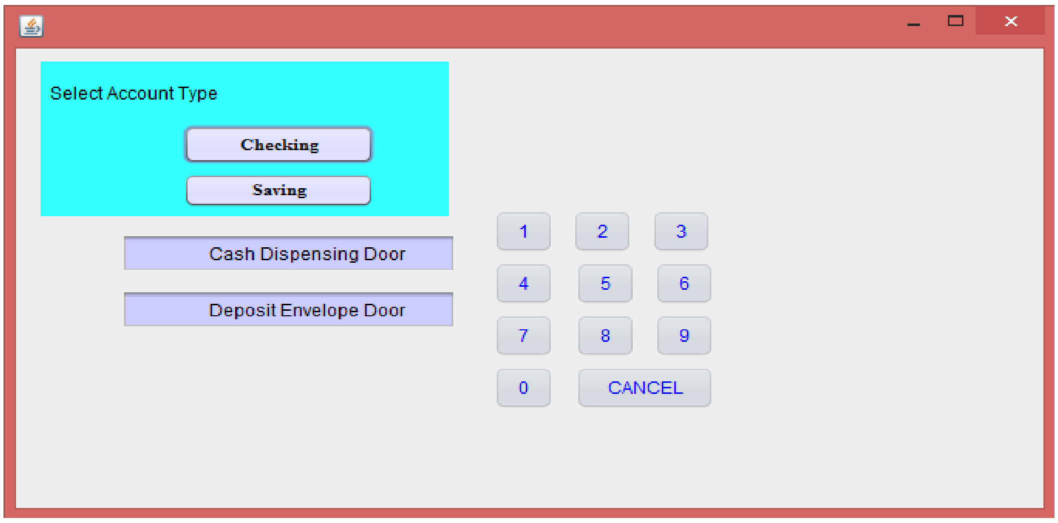


- If entered PIN is correct he will be taken to the next screen.

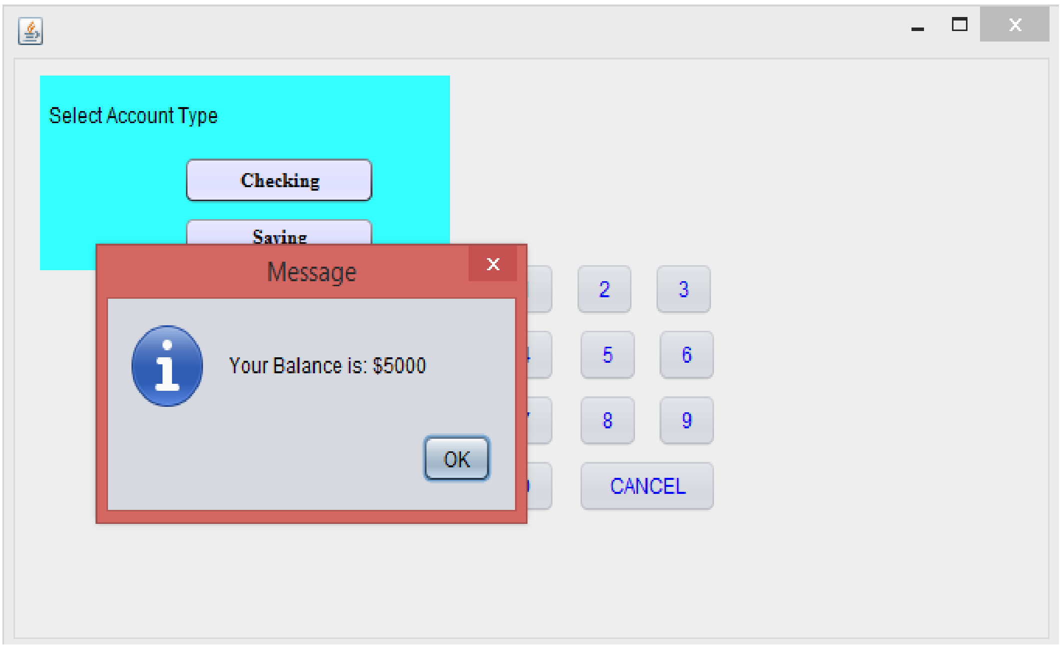


Hello BASAVAPRASAD CHANDU! Select Transaction Type

- If the customer clicks on Balance option: he will be asked to select the account displayed below.



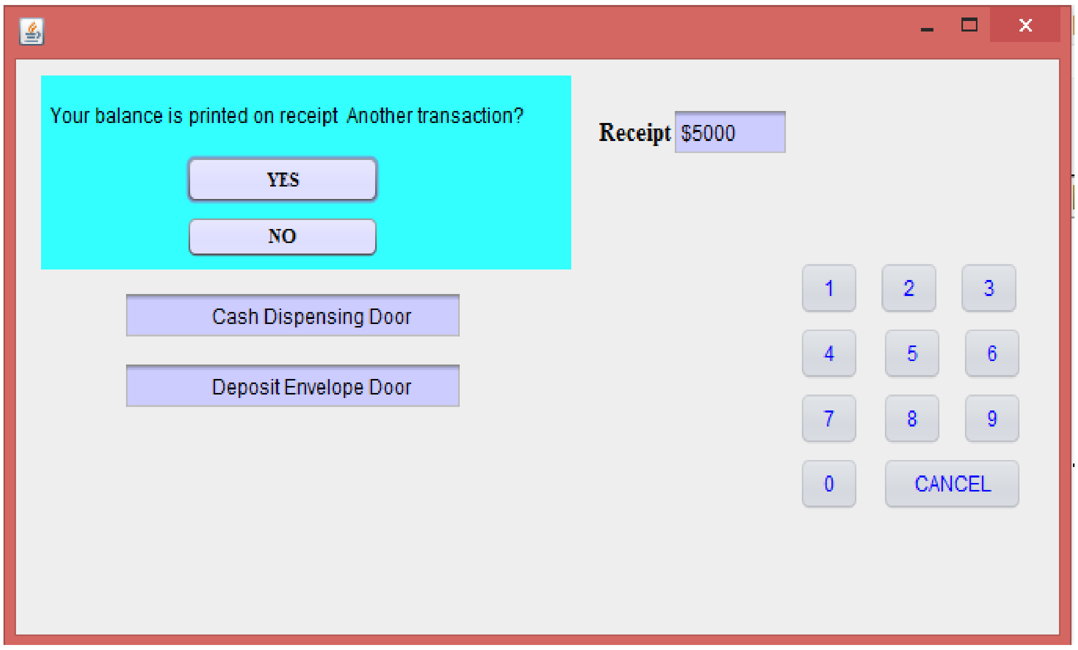
➢ Whatever the account user selects, he will be shown balance of that account which can be seen in below screen.



Message

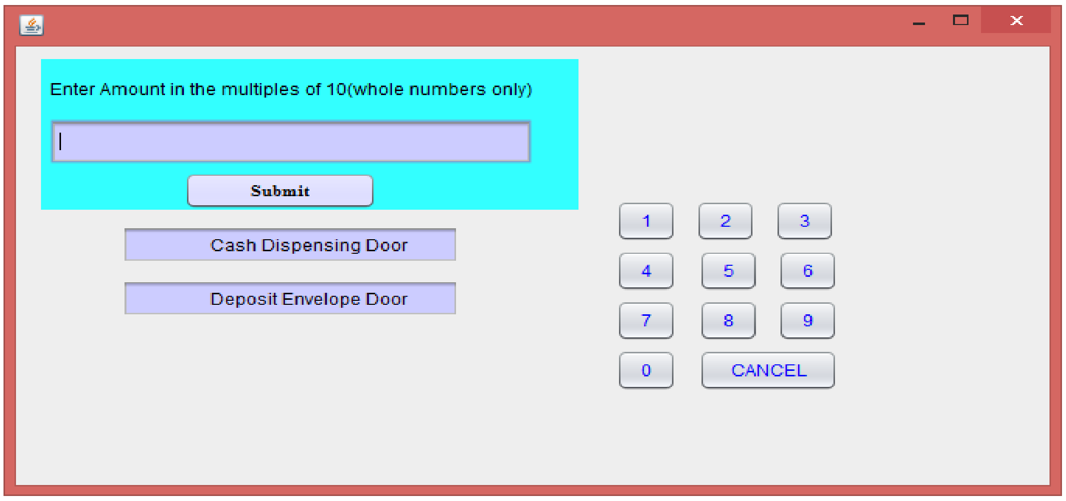
Your Balance is: $5000

➢ After, this customer will be asked that if he want to do another transaction.

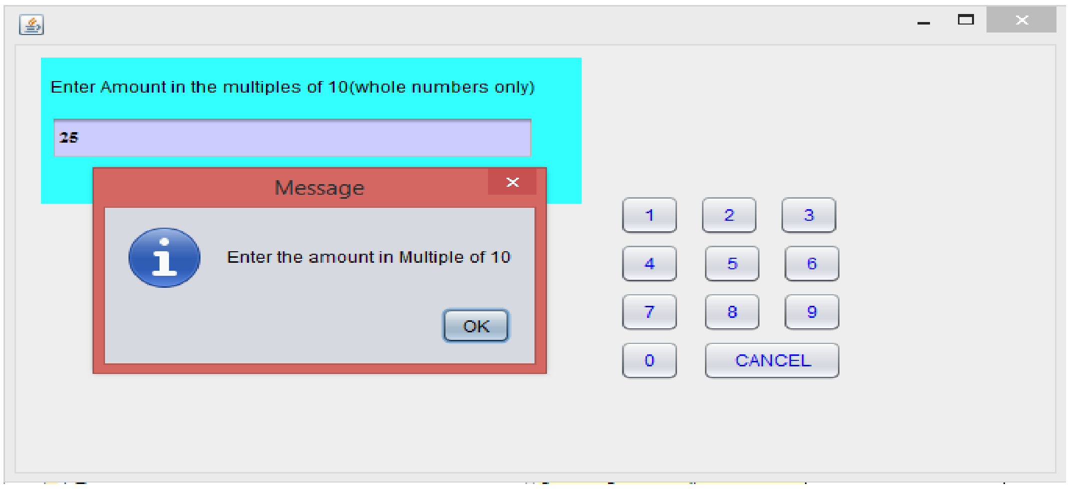

Your balance is printed on receipt Another transaction?

Receipt $5000

➢ If he selects yes, customer will be shown screen where he will asked the transaction he/she wants to do, like balance, deposit and withdrawal. If customer selects withdrawal, following screen will be displayed.


Enter Amount in the multiples of 10(whole numbers only)

➢ If the entered amount is not in multiple of 10, following screen will comes up.

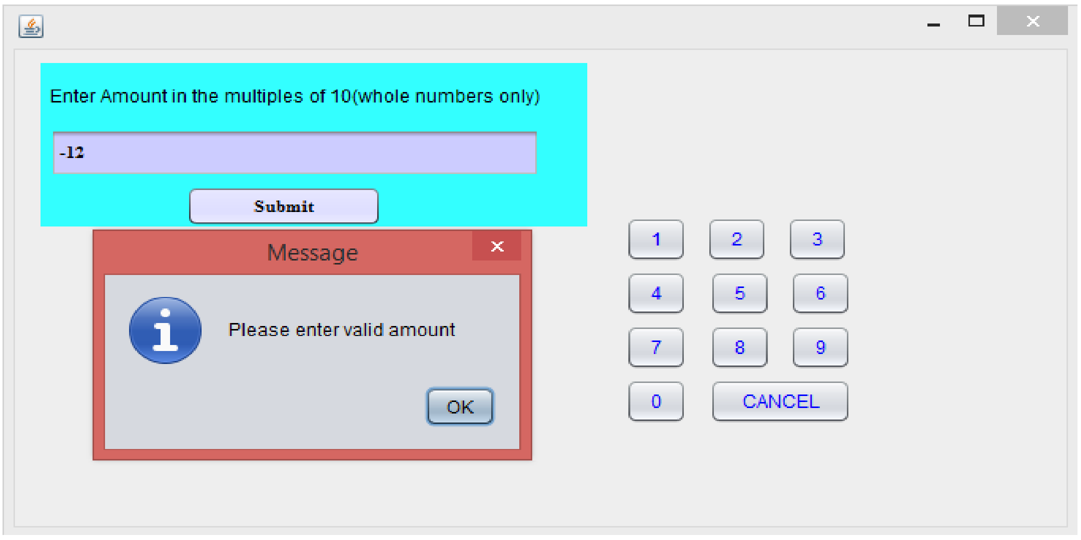

Enter Amount in the multiples of 10(whole numbers only)

25

Message

Enter the amount in Multiple of 10

➢ If the entered amount is invalid, following screen will come up.

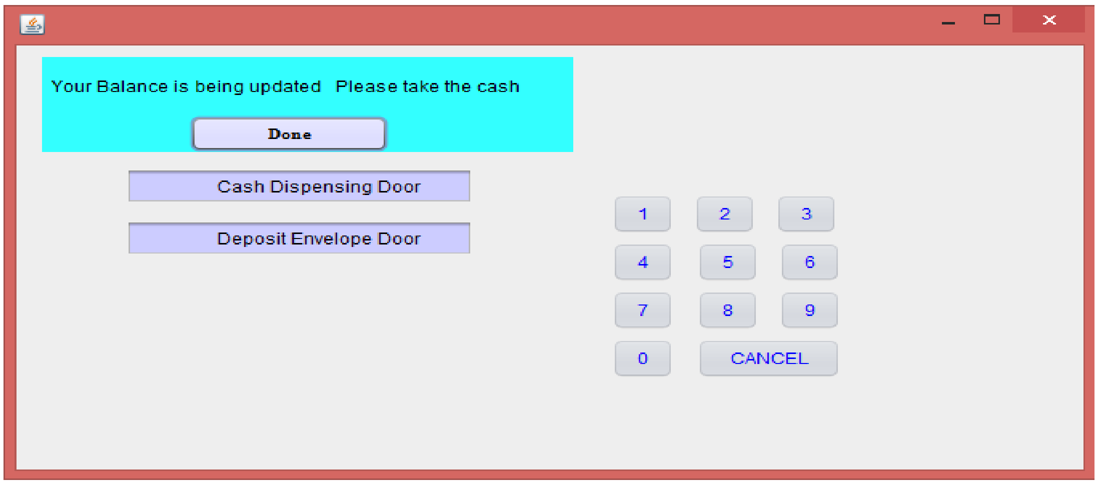

Enter Amount in the multiples of 10(whole numbers only)

-12

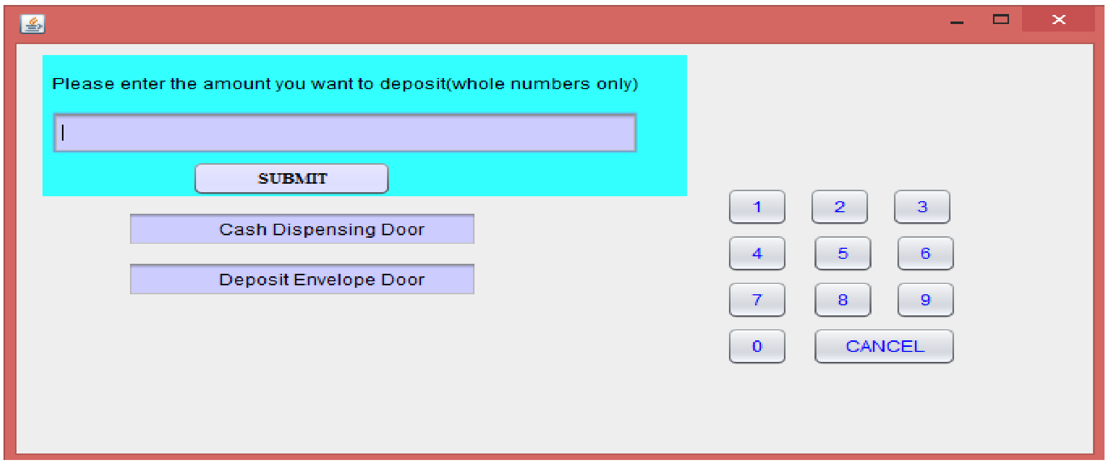
Message

Please enter valid amount

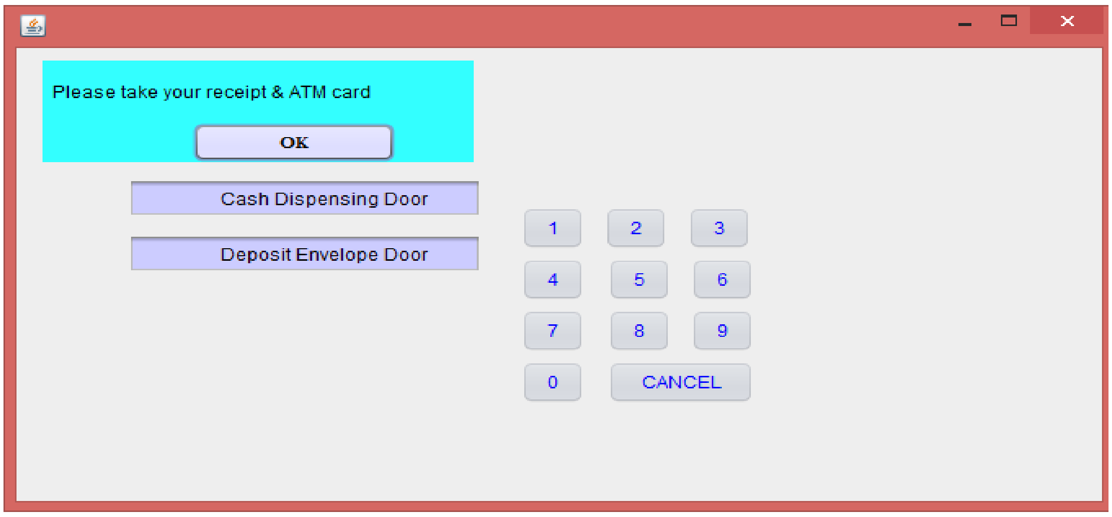
➢ If he enters the correct amount, following screen will come up.


Your Balance is being updated Please take the cash

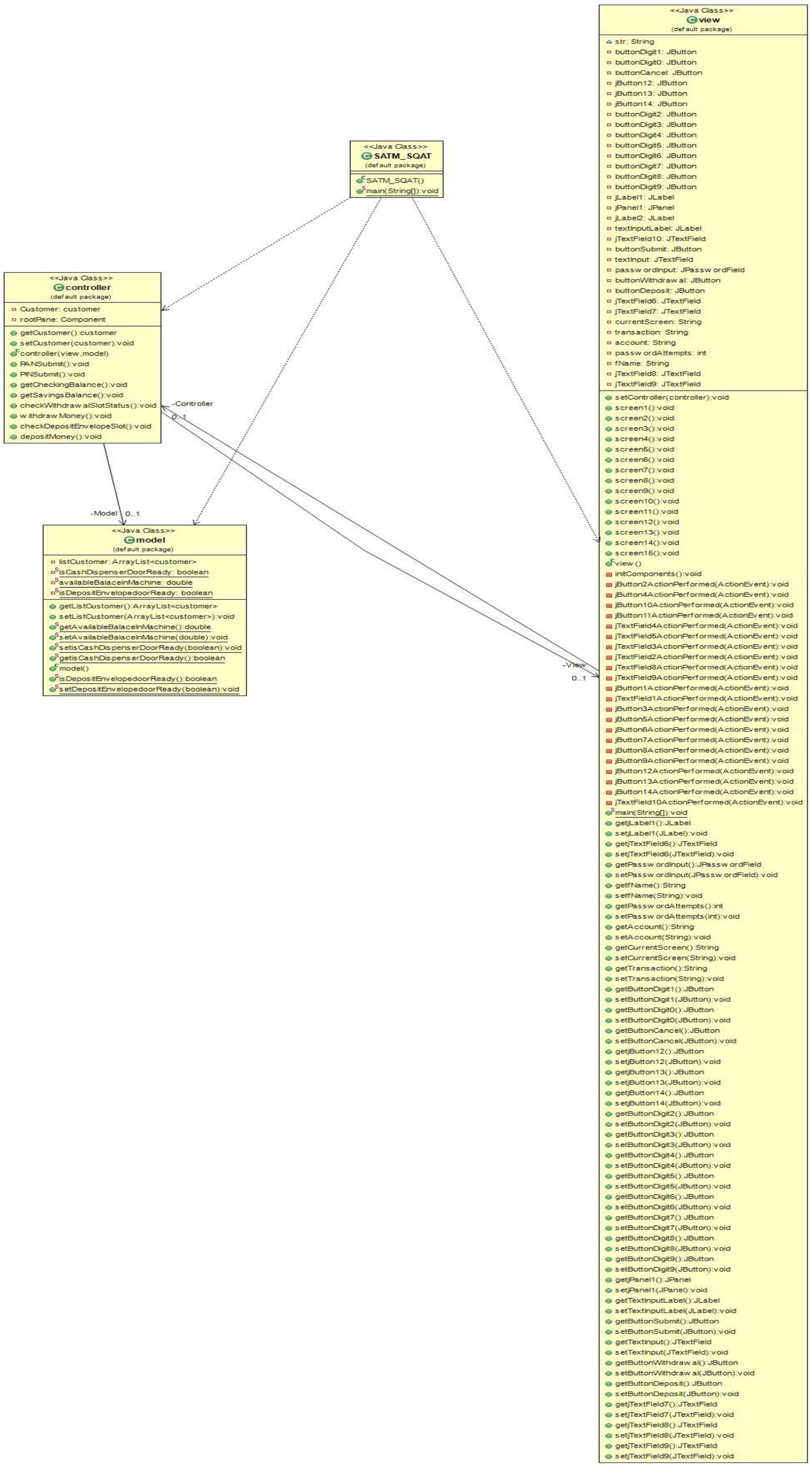
- On Pressing done button, customer will be shown screen where he will asked the transaction he/she wants to do, like balance, deposit and withdrawal. If customer selects balance, following screen will be displayed.


Please enter the amount you want to deposit(whole numbers only)
SUBMIT
Cash Dispensing Door
Deposit Envelope Door
1
2
3
4
5
6
7
8
9
0
CANCEL

- On entering correct amount, customer will be shown screen where he will asked if he want to do another transaction, if he says no. following screen will come.



CLASS DIAGRAM OF MVC ARCHITECTURE:



<<Java Class>>
SATM_SQAT
(default package)
SATM_SQAT()
main(String[]):void
<<Java Class>>
controller
(default package)
Customer: customer
rootPane: Component
getCustomer():customer
setCustomer(customer):void
controller(view,model)
PANSubmit():void
PINSubmit():void
getCheckingBalance():void
getSavingsBalance():void
checkWithdrawalSlotStatus():void
withdrawMoney():void
checkDepositEnvelopeSlot():void
depositMoney():void
-Controller
0..1
-Model
0..1
<<Java Class>>
model
(default package)
listCustomer: ArrayList<customer>
isCashDispenserDoorReady: boolean
availableBalaceInMachine: double
isDepositEnvelopedoorReady: boolean
getListCustomer():ArrayList<customer>
setListCustomer(ArrayList<customer>):void
getAvailableBalaceInMachine():double
setAvailableBalaceInMachine(double):void
setisCashDispenserDoorReady(boolean):void
getisCashDispenserDoorReady():boolean
model()
isDepositEnvelopedoorReady():boolean
setDepositEnvelopedoorReady(boolean):void
-View
0..1
<<Java Class>>
view
(default package)
str: String
buttonDigit1: JButton
buttonDigit0: JButton
buttonCancel: JButton
jButton12: JButton
jButton13: JButton
jButton14: JButton
buttonDigit2: JButton
buttonDigit3: JButton
buttonDigit4: JButton
buttonDigit5: JButton
buttonDigit6: JButton
buttonDigit7: JButton
buttonDigit8: JButton
buttonDigit9: JButton
jLabel1: JLabel
jPanel1: JPanel
jLabel2: JLabel
textInputLabel: JLabel
jTextField10: JTextField
buttonSubmit: JButton
textInput: JTextField
passwordInput: JPasswordField
buttonWithdrawal: JButton
buttonDeposit: JButton
jTextField6: JTextField
jTextField7: JTextField
currentScreen: String
transaction: String
account: String
passwordAttempts: int
fName: String
jTextField8: JTextField
jTextField9: JTextField
setController(controller):void
screen1():void
screen2():void
screen3():void
screen4():void
screen5():void
screen6():void
screen7():void
screen8():void
screen9():void
screen10():void
screen11():void
screen12():void
screen13():void
screen14():void
screen15():void
view()
initComponents():void
jButton2ActionPerformed(ActionEvent):void
jButton4ActionPerformed(ActionEvent):void
jButton10ActionPerformed(ActionEvent):void
jButton11ActionPerformed(ActionEvent):void
jTextField4ActionPerformed(ActionEvent):void
jTextField5ActionPerformed(ActionEvent):void
jTextField3ActionPerformed(ActionEvent):void
jTextField2ActionPerformed(ActionEvent):void
jTextField8ActionPerformed(ActionEvent):void
jTextField9ActionPerformed(ActionEvent):void
jButton1ActionPerformed(ActionEvent):void
jTextField1ActionPerformed(ActionEvent):void
jButton3ActionPerformed(ActionEvent):void
jButton5ActionPerformed(ActionEvent):void
jButton6ActionPerformed(ActionEvent):void
jButton7ActionPerformed(ActionEvent):void
jButton8ActionPerformed(ActionEvent):void
jButton9ActionPerformed(ActionEvent):void
jButton12ActionPerformed(ActionEvent):void
jButton13ActionPerformed(ActionEvent):void
jButton14ActionPerformed(ActionEvent):void
jTextField10ActionPerformed(ActionEvent):void
main(String[]):void
getjLabel1():JLabel
setjLabel1(JLabel):void
getjTextField6():JTextField
setjTextField6(JTextField):void
getPasswordInput():JPasswordField
setPasswordInput(JPasswordField):void
getfName():String
setfName(String):void
getPasswordAttempts():int
setPasswordAttempts(int):void
getAccount():String
setAccount(String):void
getCurrentScreen():String
setCurrentScreen(String):void
getTransaction():String
setTransaction(String):void
getButtonDigit1():JButton
setButtonDigit1(JButton):void
getButtonDigit0():JButton
setButtonDigit0(JButton):void
getButtonCancel():JButton
setButtonCancel(JButton):void
getjButton12():JButton
setjButton12(JButton):void
getjButton13():JButton
setjButton13(JButton):void
getjButton14():JButton
setjButton14(JButton):void
getButtonDigit2():JButton
setButtonDigit2(JButton):void
getButtonDigit3():JButton
setButtonDigit3(JButton):void
getButtonDigit4():JButton
setButtonDigit4(JButton):void
getButtonDigit5():JButton
setButtonDigit5(JButton):void
getButtonDigit6():JButton
setButtonDigit6(JButton):void
getButtonDigit7():JButton
setButtonDigit7(JButton):void
getButtonDigit8():JButton
setButtonDigit8(JButton):void
getButtonDigit9():JButton
setButtonDigit9(JButton):void
getjPanel1():JPanel
setjPanel1(JPanel):void
getTextInputLabel():JLabel
setTextInputLabel(JLabel):void
getButtonSubmit():JButton
setButtonSubmit(JButton):void
getTextInput():JTextField
setTextInput(JTextField):void
getButtonWithdrawal():JButton
setButtonWithdrawal(JButton):void
getButtonDeposit():JButton
setButtonDeposit(JButton):void
getjTextField7():JTextField
setjTextField7(JTextField):void
getjTextField8():JTextField
setjTextField8(JTextField):void
getjTextField9():JTextField
setjTextField9(JTextField):void

The controller class acts as an interface between the view class and the model class. The model class performs the computations and then gives the result. The controller takes the result from the model class and passes it to the view class to display it on the GUI.

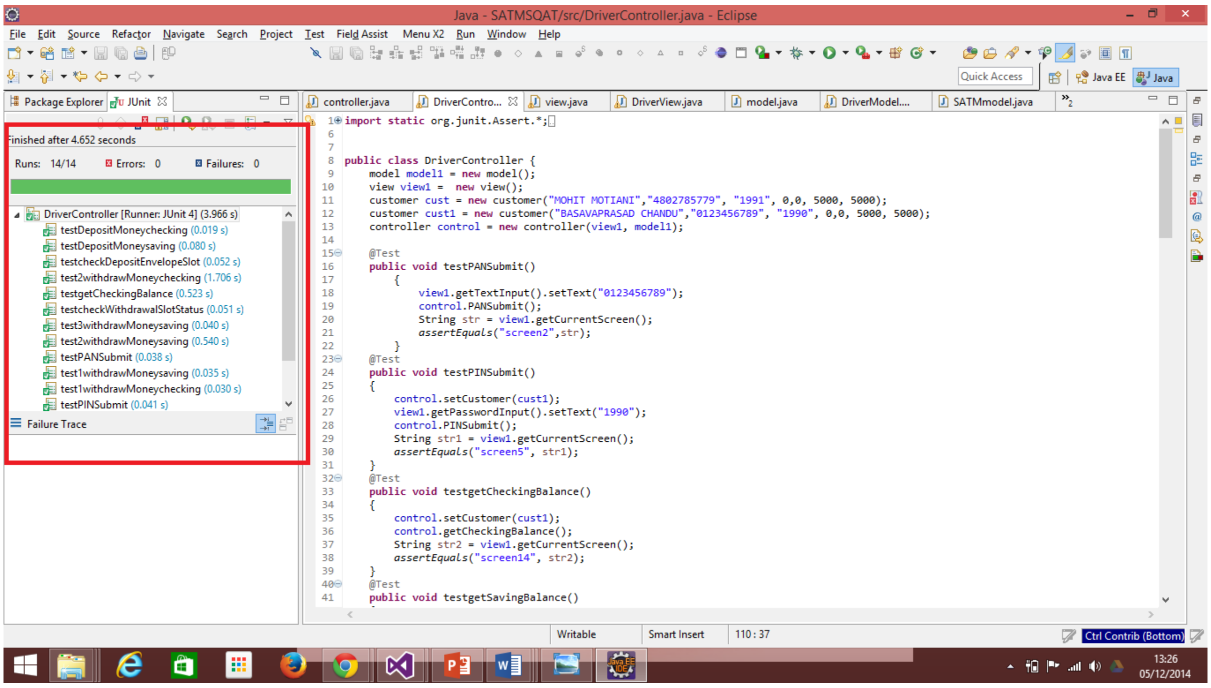
TESTING OF SATM APPLICATION

Designing test cases and implementing the test cases is the primary function of this phase. Here, we test the SATM application through the test cases. Junit test drivers are used to implement them. Since, we have implemented the SATM application using MVC architecture, we can test entire application in units or components. We start the testing at unit level, then later for integration testing. This is later surpassed by object oriented system testing.

1) **Test cases for controller:**
   Testing of controller is done using the model based testing strategy. We have different user scenarios and use cases. As discussed, the use case scenarios would be similar to that of model class and view class. However, the implementing setter and getter methods would change. The method that calculates the change will also get changed.

Test Case	Function	Input	Expected Output
1	PANSubmit()	PAN = 0123456789	Screen2
2	PINSubmit()	PIN = 1990	Screen5
3	getCheckingBalance()	Account = Checking	Screen14
4	getSavingBalance	Account = Savings	Screen14
5	checkWithdrawalSlotStatus()	getCurrentScreen()	Screen7
6	withdrwaMoneyChecking()	Account= Checking, Amount = 350	Scree11
7	withdrwaMoneyChecking()	Account= Checking, Amount = 35	Screen7
8	withdrwaMoneyChecking()	Account= Checking, Amount = 100000	Screen9
9	withdrwaMoneySaving()	Account= Saving, Amount = 350	Screen11
10	withdrwaMoneySaving()	Account= Saving, Amount = 35	Screen7
11	withdrwaMoneySaving()	Account= Saving, Amount = 100000	Screen9
12	checkDepositEnvelopeSlot()	getCurrentScreen()	Screen13

13	depositMoneyChecking()	Account= Checking, Amount = 50	Screen14
14	depositMoneySaving()	Account= Saving, Amount = 50	Screen14



```
Finished after 4.652 seconds
Runs: 14/14    Errors: 0    Failures: 0

DriverController [Runner: JUnit 4] (3.966 s)
    testDepositMoneychecking (0.019 s)
    testDepositMoneysaving (0.080 s)
    testcheckDepositEnvelopeSlot (0.052 s)
    test2withdrawMoneychecking (1.706 s)
    testgetCheckingBalance (0.523 s)
    testcheckWithdrawalSlotStatus (0.051 s)
    test3withdrawMoneysaving (0.040 s)
    test2withdrawMoneysaving (0.540 s)
    testPANSubmit (0.038 s)
    test1withdrawMoneysaving (0.035 s)
    test1withdrawMoneychecking (0.030 s)
    testPINSubmit (0.041 s)
```

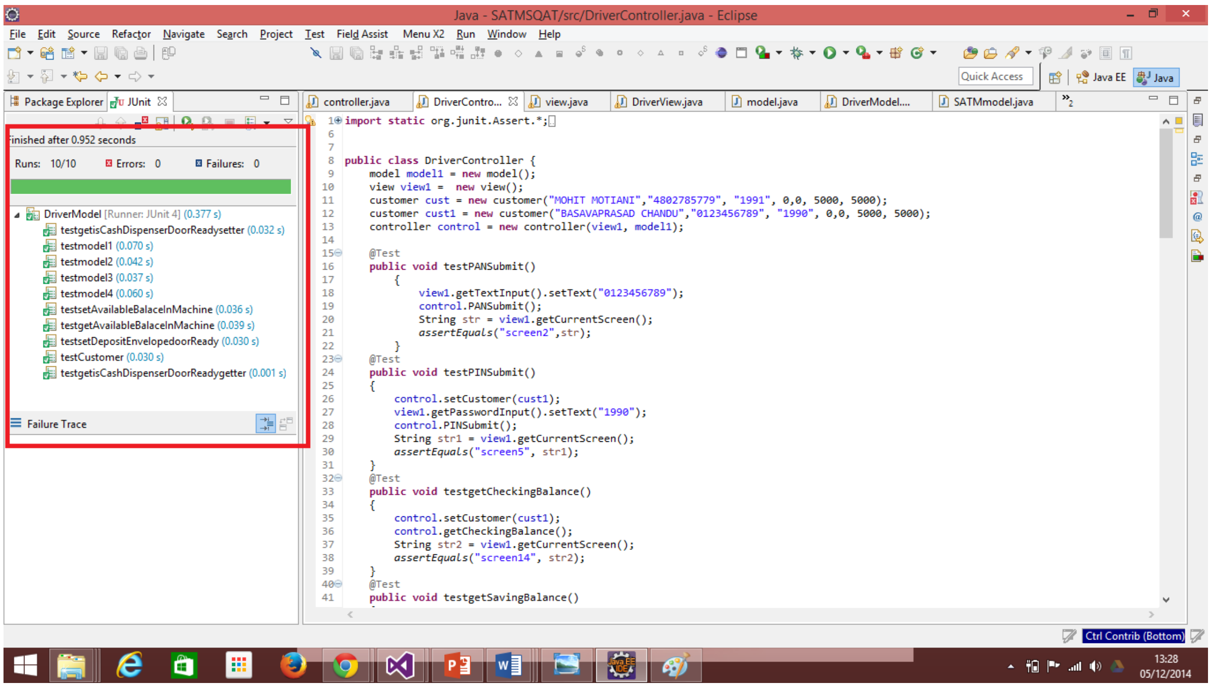
```java
import static org.junit.Assert.*;


public class DriverController {
    model model1 = new model();
    view view1 =  new view();
    customer cust = new customer("MOHIT MOTIANI","4802785779", "1991", 0,0, 5000, 5000);
    customer cust1 = new customer("BASAVAPRASAD CHANDU","0123456789", "1990", 0,0, 5000, 5000);
    controller control = new controller(view1, model1);

    @Test
    public void testPANSubmit()
        {
            view1.getTextInput().setText("0123456789");
            control.PANSubmit();
            String str = view1.getCurrentScreen();
            assertEquals("screen2",str);
        }
    @Test
    public void testPINSubmit()
    {
        control.setCustomer(cust1);
        view1.getPasswordInput().setText("1990");
        control.PINSubmit();
        String str1 = view1.getCurrentScreen();
        assertEquals("screen5", str1);
    }
    @Test
    public void testgetCheckingBalance()
    {
        control.setCustomer(cust1);
        control.getCheckingBalance();
        String str2 = view1.getCurrentScreen();
        assertEquals("screen14", str2);
    }
    @Test
    public void testgetSavingBalance()
```

2) **Test cases for model**:

We can design the test cases for "Model" class by testing the methods within the class along using the functionality of the model in the driver. The class "Model" has methods which get invoked when respective buttons are clicked. There are number of test cases to test the model class of the SATM application. Following are the test cases of Model class.

Test Case	**Function**	**Input**	**Expected Output**
1	Model1()	getCustomerList().get(0)	BASAVAPRASAD CHANDU, 0123456789, 1990, 0, 0, 5000, 5000
2	Model2()	getCustomerList().get(1)	MOHIT MOTIANI, 4802785779, 1991, 0, 0, 5000, 5000
3	Model3()	getCustomerList().get(2)	SAI SASHANK GOPARAJU, 1234567890, 1989, 0, 0, 5000, 5000

4	Model4()	getCustomerList().get(3)	CHHATRAPAL SISODIYA, 9876543210, 1992, 0, 0, 5000, 5000
5	getisCashDispenserDoorReadygetter()	getisCashDispenserDoorReady()	True
6	getisCashDispenserDoorReadysetter()	getisCashDispenserDoorReady()	True
7	getAvailableBalanceInMachine()	getAvailableBalanceInMachine()	100000
8	setAvailableBalanceInMachine()	getAvailableBalanceInMachine()	100000
9	setDepositEnvelopdoorReady()	isDepositEnvelopedoorReady()	True



```
Finished after 0.952 seconds
Runs: 10/10    Errors: 0    Failures: 0

DriverModel [Runner: JUnit 4] (0.377 s)
    testgetisCashDispenserDoorReadysetter (0.032 s)
    testmodel1 (0.070 s)
    testmodel2 (0.042 s)
    testmodel3 (0.037 s)
    testmodel4 (0.060 s)
    testsetAvailableBalaceInMachine (0.036 s)
    testgetAvailableBalaceInMachine (0.039 s)
    testsetDepositEnvelopedoorReady (0.030 s)
    testCustomer (0.030 s)
    testgetisCashDispenserDoorReadygetter (0.001 s)
```

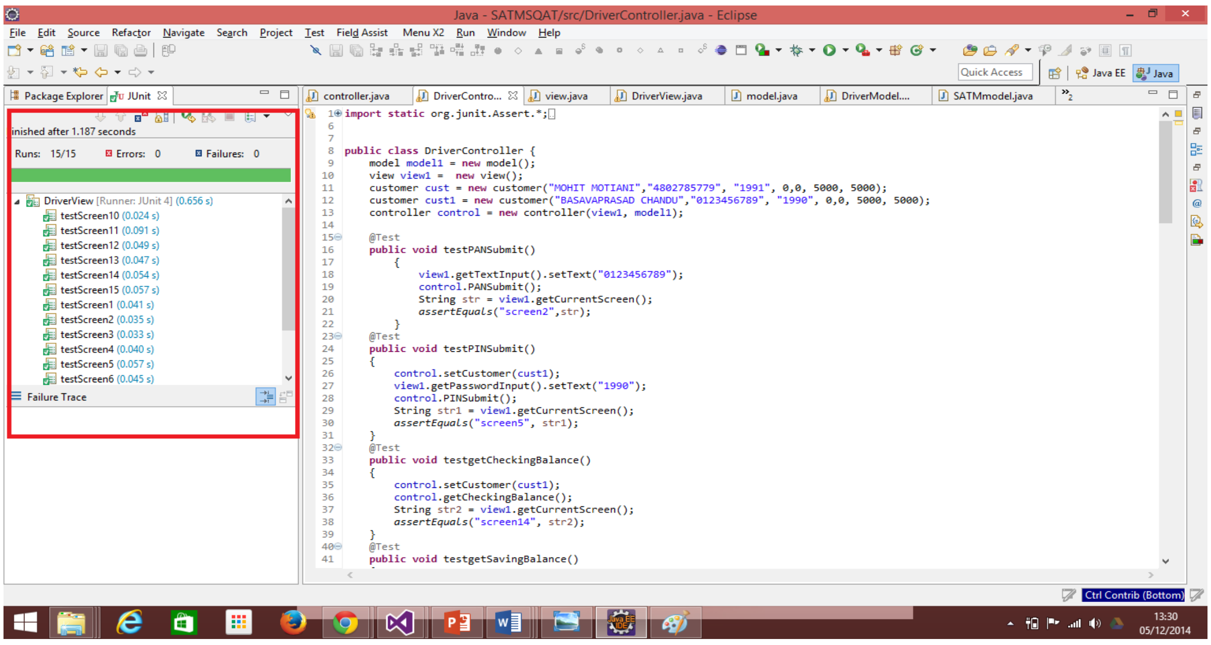
```java
import static org.junit.Assert.*;


public class DriverController {
    model model1 = new model();
    view view1 =  new view();
    customer cust = new customer("MOHIT MOTIANI","4802785779", "1991", 0,0, 5000, 5000);
    customer cust1 = new customer("BASAVAPRASAD CHANDU","0123456789", "1990", 0,0, 5000, 5000);
    controller control = new controller(view1, model1);

    @Test
    public void testPANSubmit()
        {
            view1.getTextInput().setText("0123456789");
            control.PANSubmit();
            String str = view1.getCurrentScreen();
            assertEquals("screen2",str);
        }
    @Test
    public void testPINSubmit()
    {
        control.setCustomer(cust1);
        view1.getPasswordInput().setText("1990");
        control.PINSubmit();
        String str1 = view1.getCurrentScreen();
        assertEquals("screen5", str1);
    }
    @Test
    public void testgetCheckingBalance()
    {
        control.setCustomer(cust1);
        control.getCheckingBalance();
        String str2 = view1.getCurrentScreen();
        assertEquals("screen14", str2);
    }
    @Test
    public void testgetSavingBalance()
```

3) Test cases for view:

Testing of VIEW is also done using model based testing strategy. However, the implementing setter and getter methods would change. The method that calculates the change will also get changed.

Test Case	Function	Input	Expected Output
1	Screen1()	getCurrentScreen()	currentScreen=1

2	Screen2()	getCurrentScreen()	currentScreen=2
3	Screen3()	getCurrentScreen()	currentScreen=3
4	Screen4()	getCurrentScreen()	currentScreen=4
5	Screen5()	getCurrentScreen()	currentScreen=5
6	Screen6()	getCurrentScreen()	currentScreen=6
7	Screen7()	getCurrentScreen()	currentScreen=7
8	Screen8()	getCurrentScreen()	currentScreen=8
9	Screen9()	getCurrentScreen()	currentScreen=9
10	Screen10()	getCurrentScreen()	currentScreen=10
11	Screen11()	getCurrentScreen()	currentScreen=11
12	Screen12()	getCurrentScreen()	currentScreen=12
13	Screen13()	getCurrentScreen()	currentScreen=13
14	Screen14()	getCurrentScreen()	currentScreen=14
15	Screen15()	getCurrentScreen()	currentScreen=15

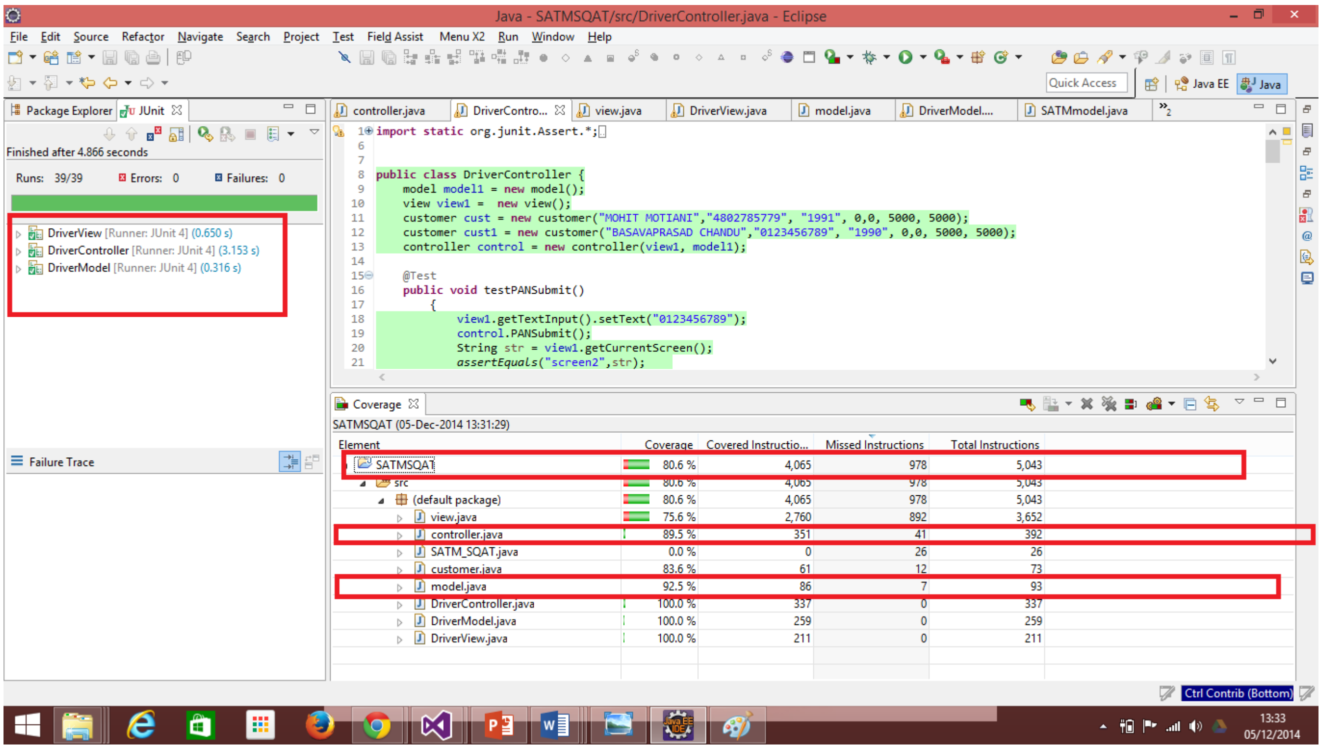


```
inished after 1.187 seconds
Runs:  15/15     Errors:  0     Failures:  0

DriverView [Runner: JUnit 4] (0.656 s)
    testScreen10 (0.024 s)
    testScreen11 (0.091 s)
    testScreen12 (0.049 s)
    testScreen13 (0.047 s)
    testScreen14 (0.054 s)
    testScreen15 (0.057 s)
    testScreen1 (0.041 s)
    testScreen2 (0.035 s)
    testScreen3 (0.033 s)
    testScreen4 (0.040 s)
    testScreen5 (0.057 s)
    testScreen6 (0.045 s)
```

```java
import static org.junit.Assert.*;


public class DriverController {
    model model1 = new model();
    view view1 =  new view();
    customer cust = new customer("MOHIT MOTIANI","4802785779", "1991", 0,0, 5000, 5000);
    customer cust1 = new customer("BASAVAPRASAD CHANDU","0123456789", "1990", 0,0, 5000, 5000);
    controller control = new controller(view1, model1);

    @Test
    public void testPANSubmit()
        {
            view1.getTextInput().setText("0123456789");
            control.PANSubmit();
            String str = view1.getCurrentScreen();
            assertEquals("screen2",str);
        }
    @Test
    public void testPINSubmit()
    {
        control.setCustomer(cust1);
        view1.getPasswordInput().setText("1990");
        control.PINSubmit();
        String str1 = view1.getCurrentScreen();
        assertEquals("screen5", str1);
    }
    @Test
    public void testgetCheckingBalance()
    {
        control.setCustomer(cust1);
        control.getCheckingBalance();
        String str2 = view1.getCurrentScreen();
        assertEquals("screen14", str2);
    }
    @Test
    public void testgetSavingBalance()
```

CODE COVERAGE:

Code coverage is a measure used to describe the degree to which the source code of a program is tested by a particular test suite. EclEmma is used in eclipse to calculate the coverage. The test cases of my SATM application covers the 81% of code in which it covers around 90% of the controller and 93% of the model. This thing can be seen in the screenshot attached below.



Finished after 4.866 seconds

Runs: 39/39 Errors: 0 Failures: 0

- DriverView [Runner: JUnit 4] (0.650 s)
- DriverController [Runner: JUnit 4] (3.153 s)
- DriverModel [Runner: JUnit 4] (0.316 s)

```java
import static org.junit.Assert.*;


public class DriverController {
    model model1 = new model();
    view view1 =  new view();
    customer cust = new customer("MOHIT MOTIANI","4802785779", "1991", 0,0, 5000, 5000);
    customer cust1 = new customer("BASAVAPRASAD CHANDU","0123456789", "1990", 0,0, 5000, 5000);
    controller control = new controller(view1, model1);

    @Test
    public void testPANSubmit()
        {
            view1.getTextInput().setText("0123456789");
            control.PANSubmit();
            String str = view1.getCurrentScreen();
            assertEquals("screen2",str);
```

SATMSQAT (05-Dec-2014 13:31:29)

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
SATMSQAT	80.6 %	4,065	978	5,043
src	80.6 %	4,065	978	5,043
(default package)	80.6 %	4,065	978	5,043
view.java	75.6 %	2,760	892	3,652
controller.java	89.5 %	351	41	392
SATM_SQAT.java	0.0 %	0	26	26
customer.java	83.6 %	61	12	73
model.java	92.5 %	86	7	93
DriverController.java	100.0 %	337	0	337
DriverModel.java	100.0 %	259	0	259
DriverView.java	100.0 %	211	0	211

This completes the testing of SATM application.